Introduction to Programming with Python

Computing

Course: CSE 1100 – Introduction to

Lecturer: Lenandlar Singh

Date: December 11, 2025

So far we have used functions to…

- get user input input()
- Produce output print()
- We can convert values from one type to another using functions
  - int()
  - float()
- Create and process lists
  - list()
  - len()
- Generate random numbers
  - randrange()

Why use Functions?

Functions

What is a function?

- Sequence / block of code that performs a specific task
- Code is organized as a unit, identified by a name
- Functions are ‘called’ when needed

Why Functions?

- We can create our own functions…

Why?

- Decomposition
  - Divide and conquer
- Modularity
  - Code organisation
- Reusability
  - Code reuse
- Code Organisation and elegance in our code
- Testing of Code – easier to test units of code for correctness, errors, finding bugs, etc
- Updating Code is easier
- Scope management

The main() function

```python
def main():
    """Main entry point of the script."""
    print("This code runs when the script is executed directly.")
    # Call other functions from here if needed
    # example_function()

if __name__ == "__main__":
    main()
```

- main()
  - entry point to every python program
  - not a built-in requirement of the language itself

- `def main():` defines a regular function named `main`.
- `if __name__ == "__main__":` is the key part. The Python interpreter sets the special `__name__` variable to the string `"__main__"` when the file is run as the main program.
- The `main()` function is called only when this condition is true.

The ‘mechanics’ of functions

- Functions often take arguments
  - This is what you have to put inside the parentheses
  - For example, print “takes in” a string
    - print(“enter your name”)
  - “Takes in” is another way of saying it takes an argument, in this case a string

- Functions often return a value
  - This is what you get back when you “call” the function
  - For example, input() returns value entered …
  - int() returns an integer
  - float() returns a float

Creating New Functions

- We don’t have to rely on only the functions in the Python library
- We can have functions that do almost anything
  - Can be as simple as adding 2 to a number
- Why use functions?
  - Makes our code easier to read
  - Lets us repeat code more efficiently

Creating a New Function

- The syntax for creating a new function is:

  ```
  def functionName(arguments):
  ```

- Let’s define a function called printName that takes in a string:

  ```
  def printName(userName):
  ```

- Like with if, elif, else, and while blocks, indentation is important
  - The function is the code indented immediately following the function definition

Simple Function

- Let's create a function that does one thing, prints "CSE 1100"

```
def printWord():
    print("CSE 1100")
```

To call this function we simply use its name:

```
printWord()
```

- This function takes in no arguments and returns no values
  - All it does when we call it is print the string "UGC 1201"

The main() + user-defined function

```
def main():
    """Main entry point of the script."""
    print("This code runs when the script is executed directly.")
    # Call other functions from here if needed
    # example_function()

if __name__ == "__main__":
    main()
```

```
def printStuff():
    print("my name is lenandlar singh")

def main():
    """Main entry point of the script."""
    print("This code runs when the script is executed directly.")
    # Call other functions from here if needed
    # example_function()
    printStuff()

if __name__ == "__main__":
    main()
```

The main() + user-defined functions

```
def printWord():
    print('this is our first function')
    print(' this function is now called')

def printSomethingElse():
    print('****************')

def main():
    """Main entry point of the script."""
    print("This code runs when the script is executed directly.")
    # Call other functions from here if needed
    # example_function()
    printWord()
    printSomethingElse()
    printWord()

if __name__ == "__main__":
    main()
```

Functions: Arguments, Return Values

```
def printIncomingData(val):
    print('the value received is ', val)


printIncomingData(5)

value = input('enter a value ')
printIncomingData(value)
```

Functions: Arguments, Return Values

```
def valSquare(value):
    value = int(value) * int(value)
    print('the answer of value * value is' , value)


value = input('enter a value ')
valSquare(value)
```

Functions: Arguments, Return Values

```
def retDoubledValues(value):
    value = int(value) + int(value)
    print('the answer of value + value is' , value)
    return value


value = input('enter a value ')
v = retDoubledValues(value)
print('the value returned is ', v)
printIncomingData(v)
```

Functions: Arguments, Return Values

```
def retDoubledValues(value):
    valueDoubled = int(value) + int(value)
    print('the answer of ', value, '+', value, 'is' , valueDoubled)
    return value


value = input('enter a value ')
v = retDoubledValues(value)
print('the value returned is ', v)
printIncomingData(v)
```

Functions: Return Multiple Values {LIST}

```
def get_stats(numbers):
    total = sum(numbers)
    count = len(numbers)
    average = total / count
    return [total, count, average] # Returns a list

data_list = get_stats([10, 20, 30, 40])
print(f"Total: {data_list[0]}, Count: {data_list[1]}, Average: {data_list[2]}")
```

In Python, the `f` prefix before a string denotes a **formatted string literal**, commonly called an **f-string**. F-strings provide a concise, readable, and efficient way to embed Python expressions and variables directly within a string.

Functions: Return Multiple Values {TUPLE}

```python
def get_user_info():
    name = "Alice"
    age = 30
    location = "New York"
    return name, age, location  # Returns a tuple ('Alice', 30, 'New York')

# Unpack the returned tuple into individual variables
user_name, user_age, user_location = get_user_info()

print(f"Name: {user_name}, Age: {user_age}, Location: {user_location}")

# Or store them in a single variable (which will be a tuple)
user_data = get_user_info()
print(f"User data tuple: {user_data}")
```

A **tuple** in Python is **an ordered, immutable collection of items**. It is similar to a list, but once a tuple is created, you cannot change, add, or remove its elements.

Return Values

- Functions can take in any number of arguments
  - From 0 ->whatever you want
- But functions can return
  - one value as variable
  - many values as tuple, list, dictionary
- To return a value we use the *return* keyword
- We can return numbers, strings, and any values stored in variables

Homework

- Write a program in python that uses function(S) - the function(S) will\
  - collect a list of n numbers from the user via the keyboard,
  - calculates how many numbers are greater than the average of all the numbers
  - Output useful results

End of Lecture!!!